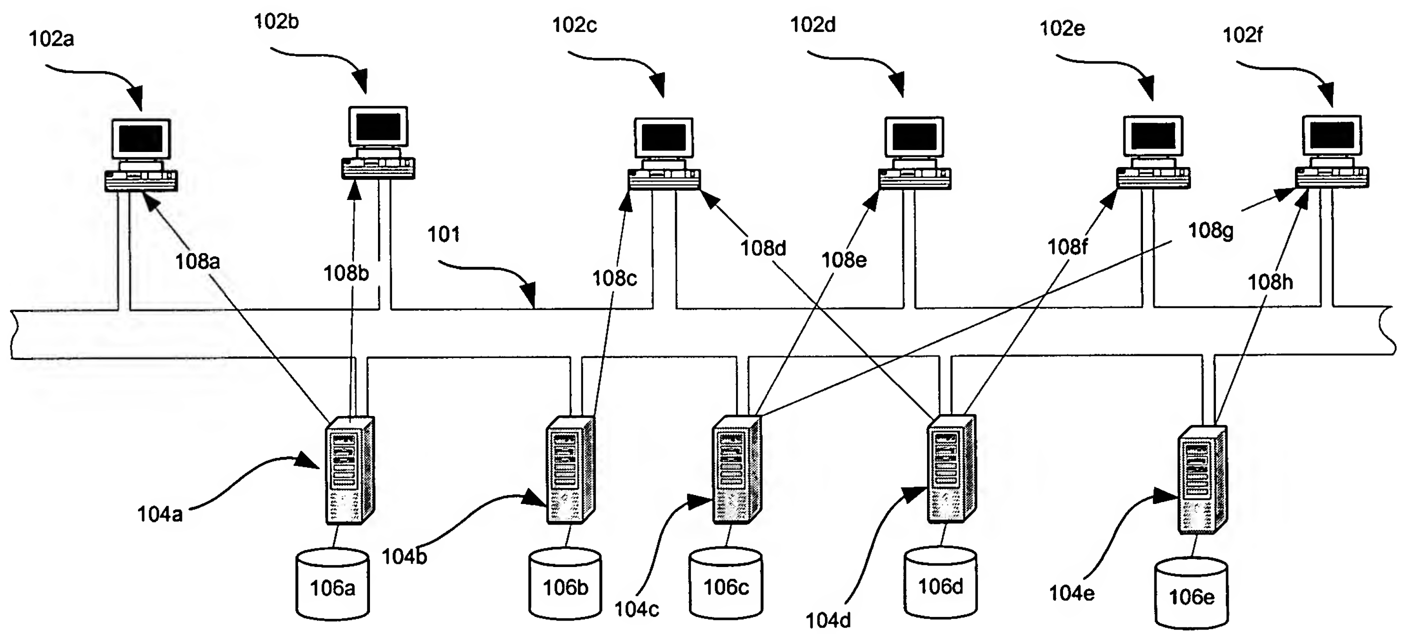

102a
102b
102c
102d
102e
102f
101
108a
108b
108c
108d
108e
108f
108g
108h
104a
104b
104c
104d
104e
106a
106b
106c
106d
106e

FIGURE 1

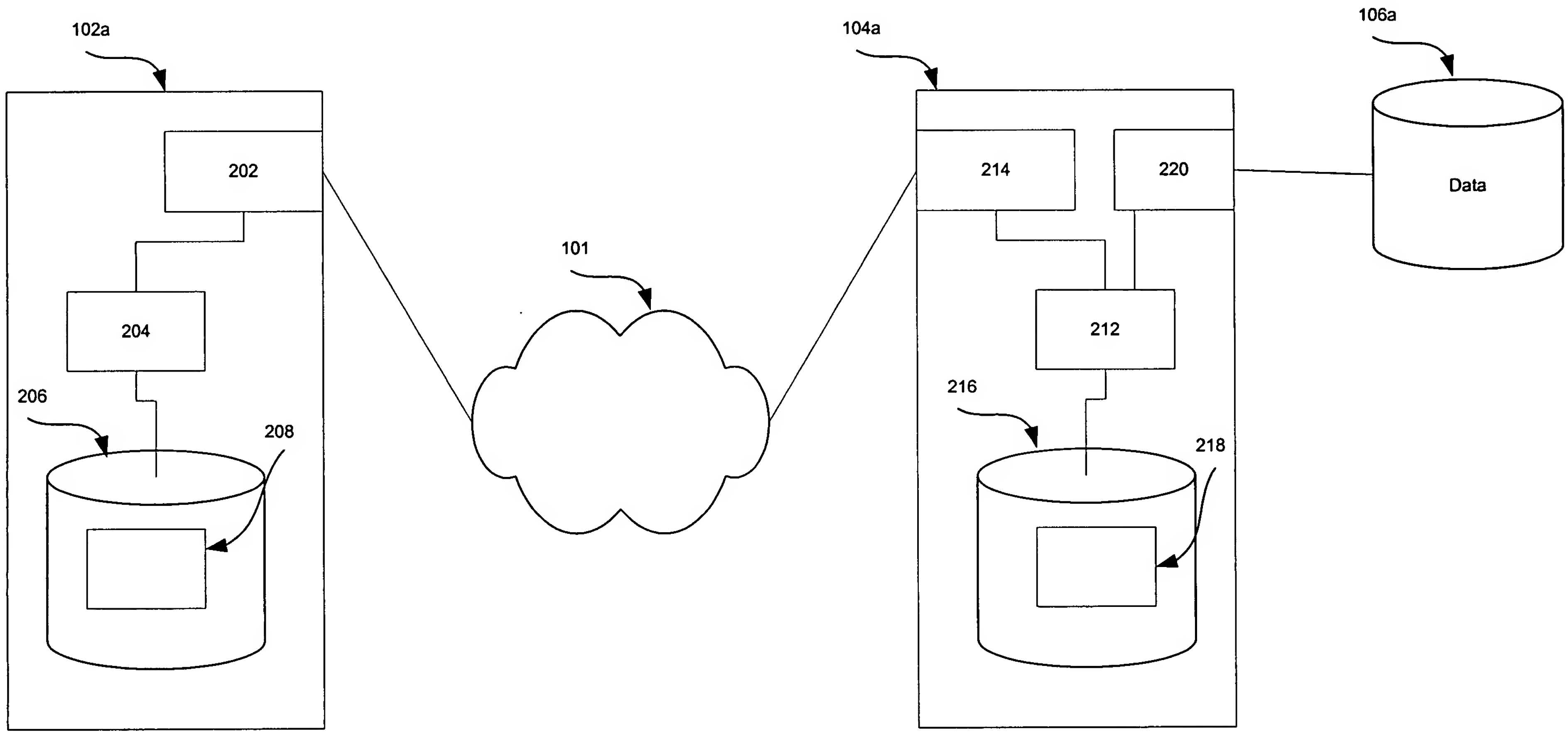


FIGURE 2

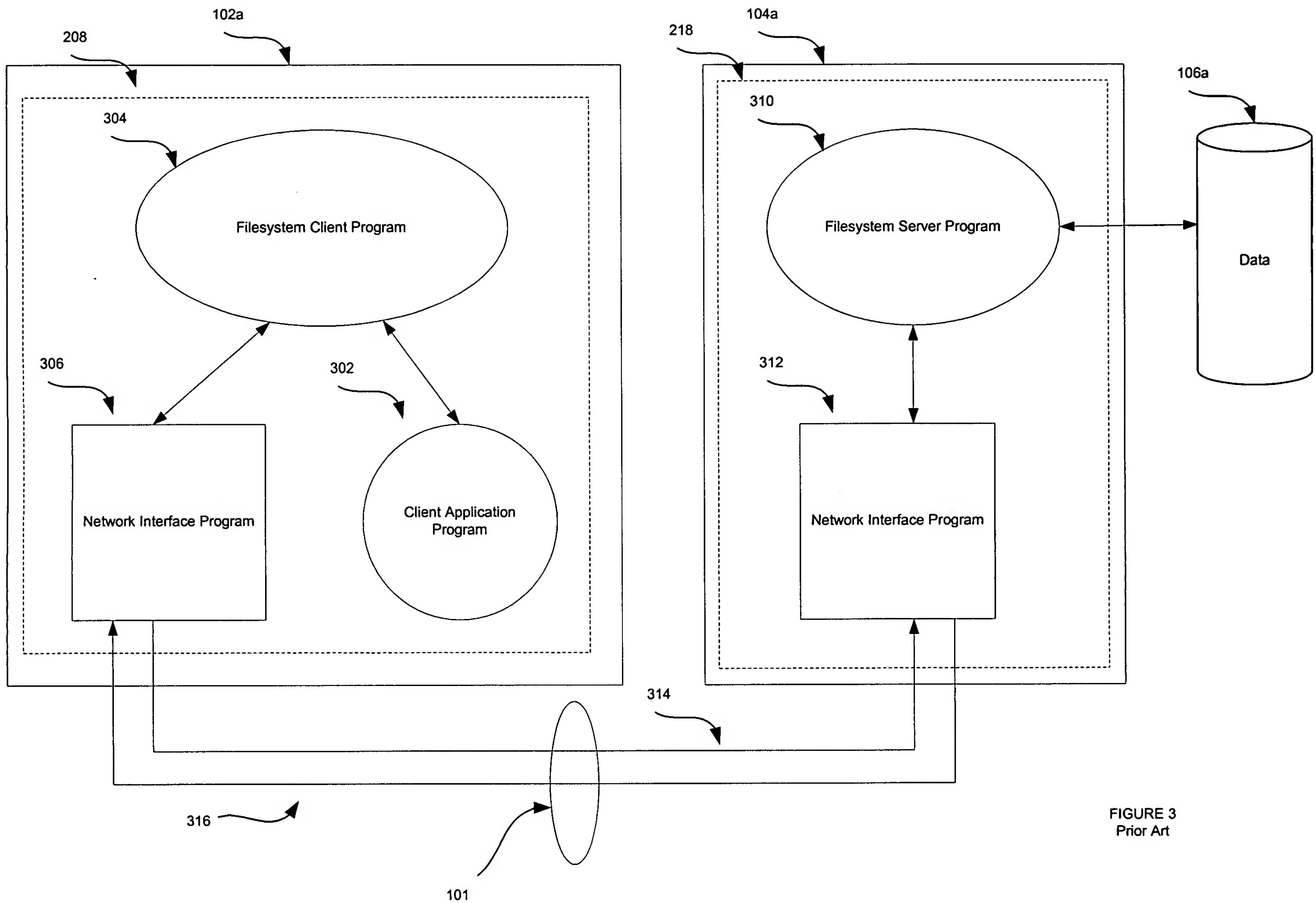


FIGURE 3
Prior Art

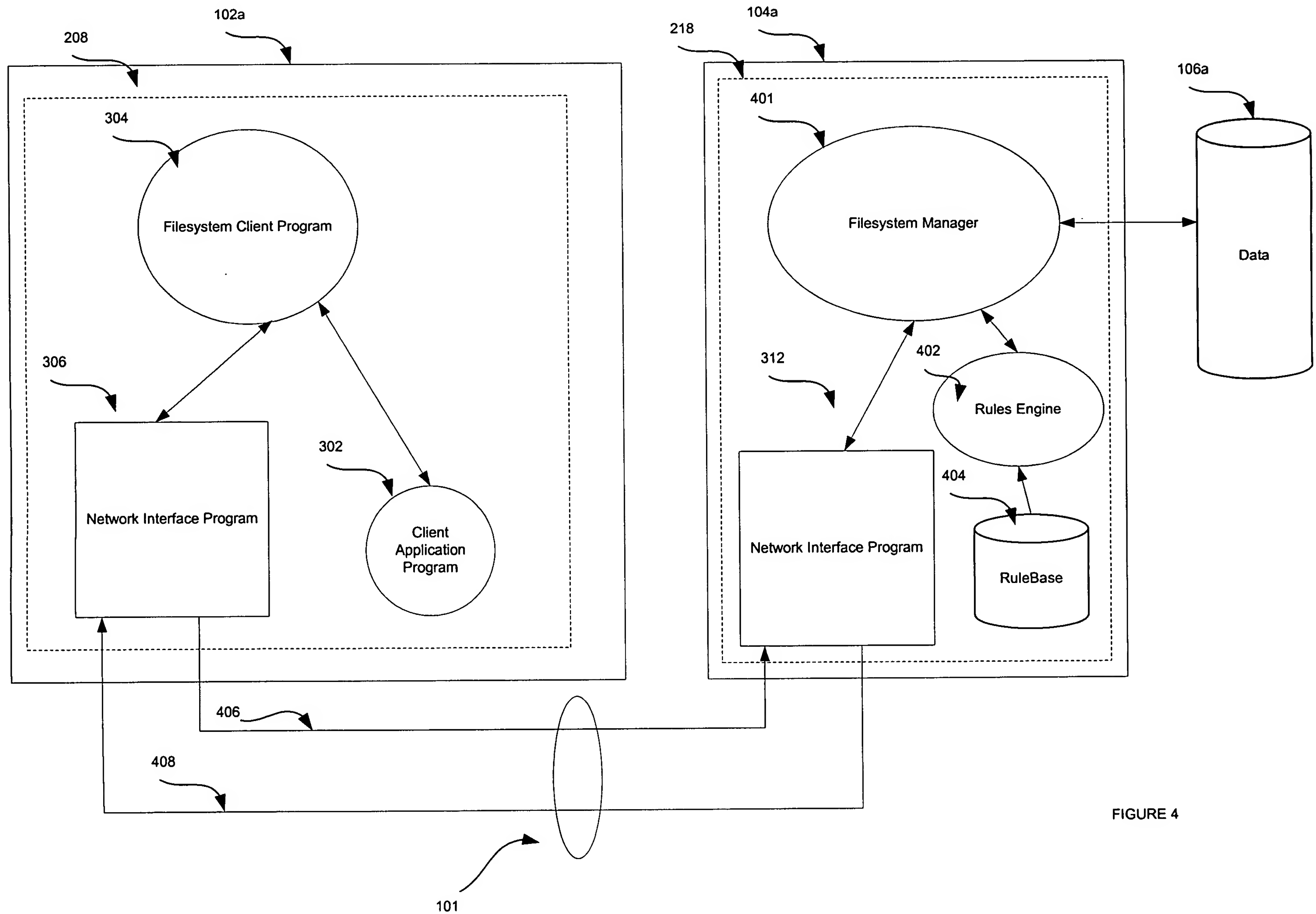

208
102a
304
Filesystem Client Program
306
Network Interface Program
302
Client
Application
Program
218
104a
401
Filesystem Manager
106a
Data
312
402
Rules Engine
404
Network Interface Program
RuleBase
406
408
101

FIGURE 4

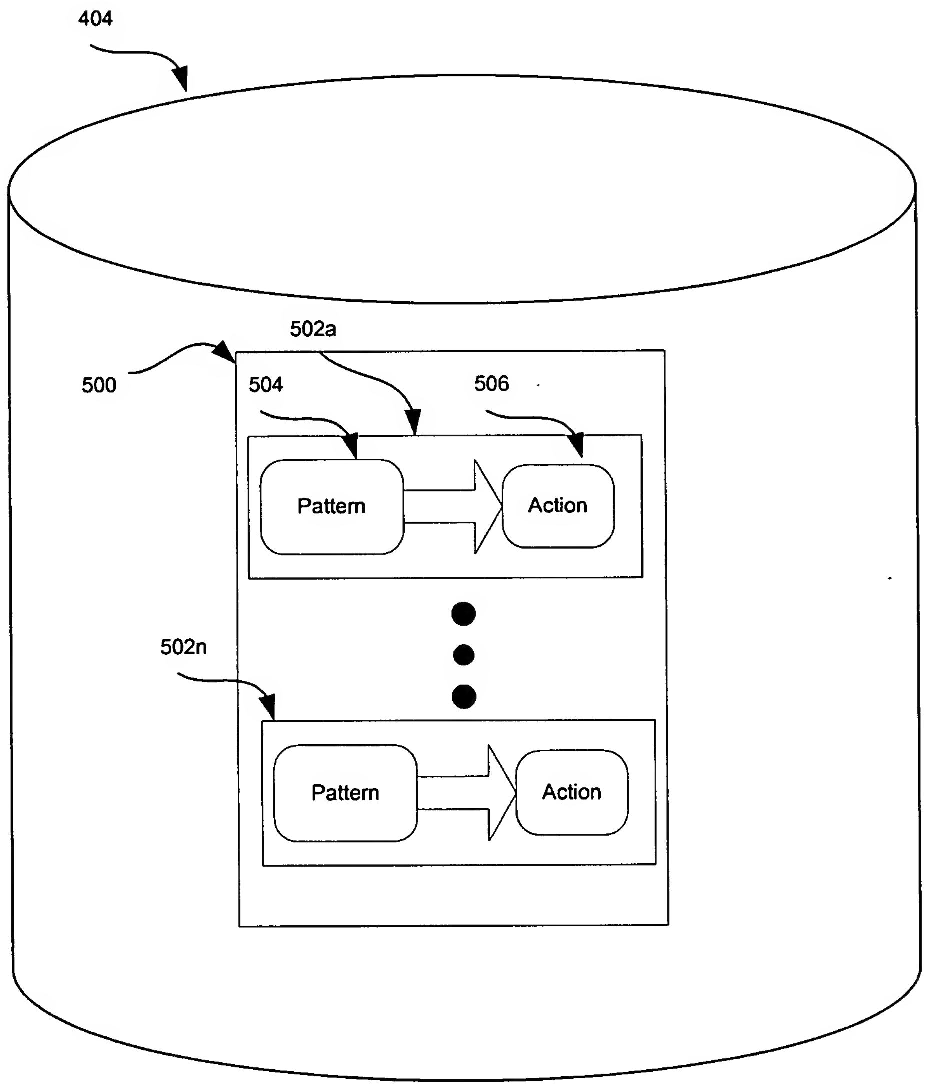


FIGURE 5

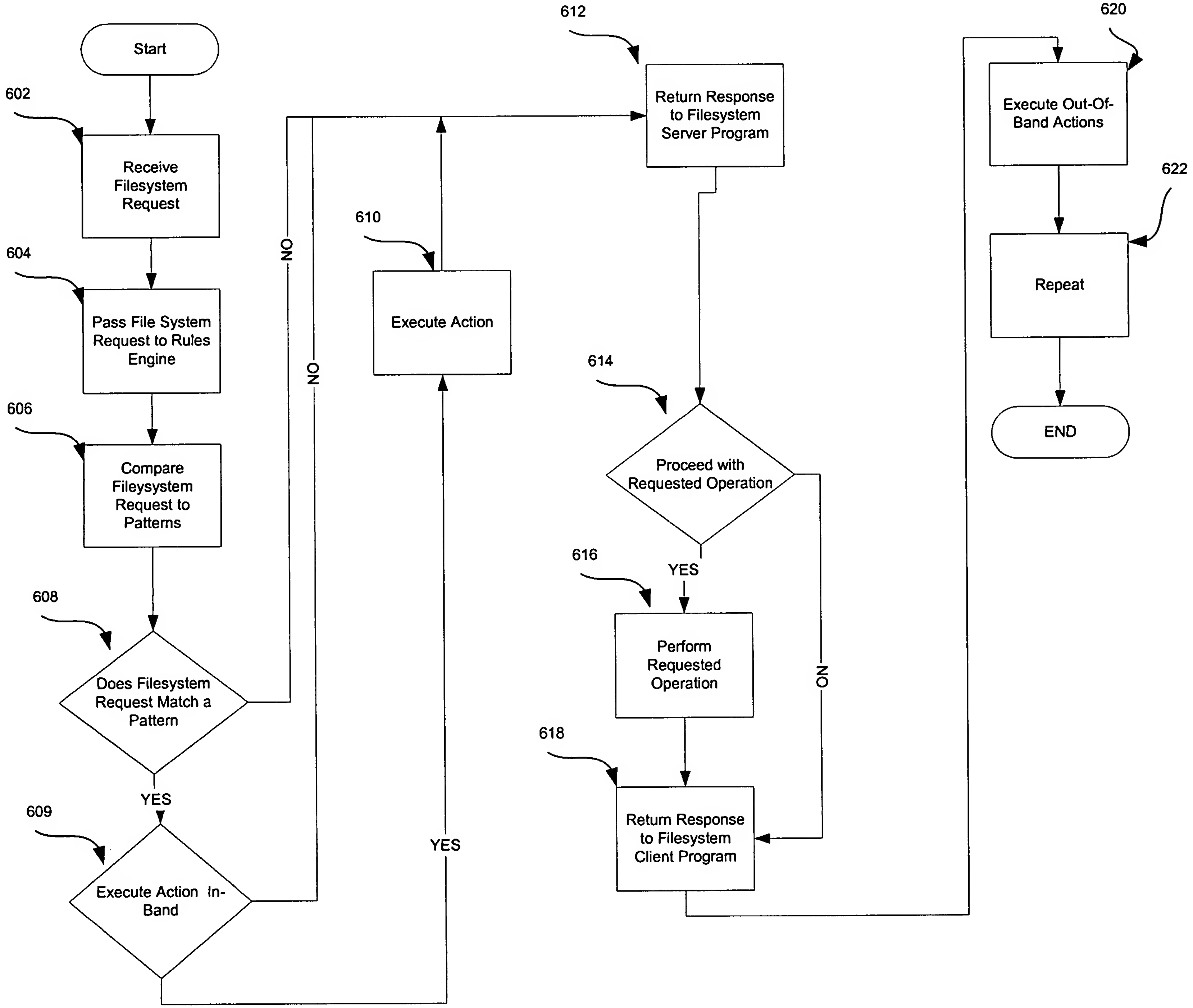

Start
602
Receive Filesystem Request
604
Pass File System Request to Rules Engine
606
Compare Fileysystem Request to Patterns
608
Does Filesystem Request Match a Pattern
YES
609
Execute Action In-Band
NO
NO
610
Execute Action
YES
612
Return Response to Filesystem Server Program
614
Proceed with Requested Operation
YES
616
Perform Requested Operation
NO
618
Return Response to Filesystem Client Program
620
Execute Out-Of-Band Actions
622
Repeat
END

FIGURE 6

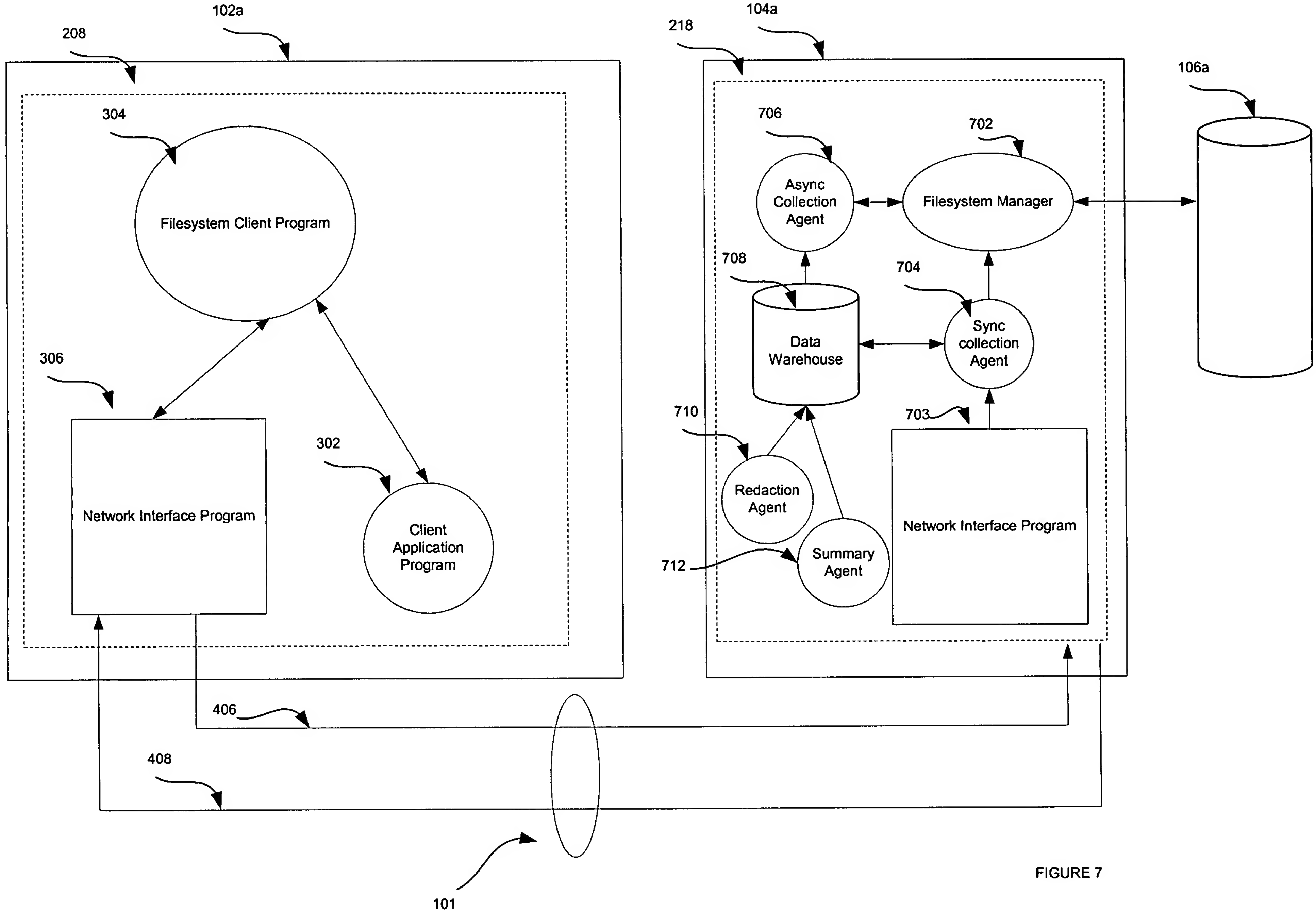

102a
208
304
Filesystem Client Program
306
Network Interface Program
302
Client
Application
Program
406
408
101
104a
218
706
Async
Collection
Agent
702
Filesystem Manager
708
Data
Warehouse
704
Sync
collection
Agent
703
710
Redaction
Agent
712
Summary
Agent
Network Interface Program
106a

FIGURE 7

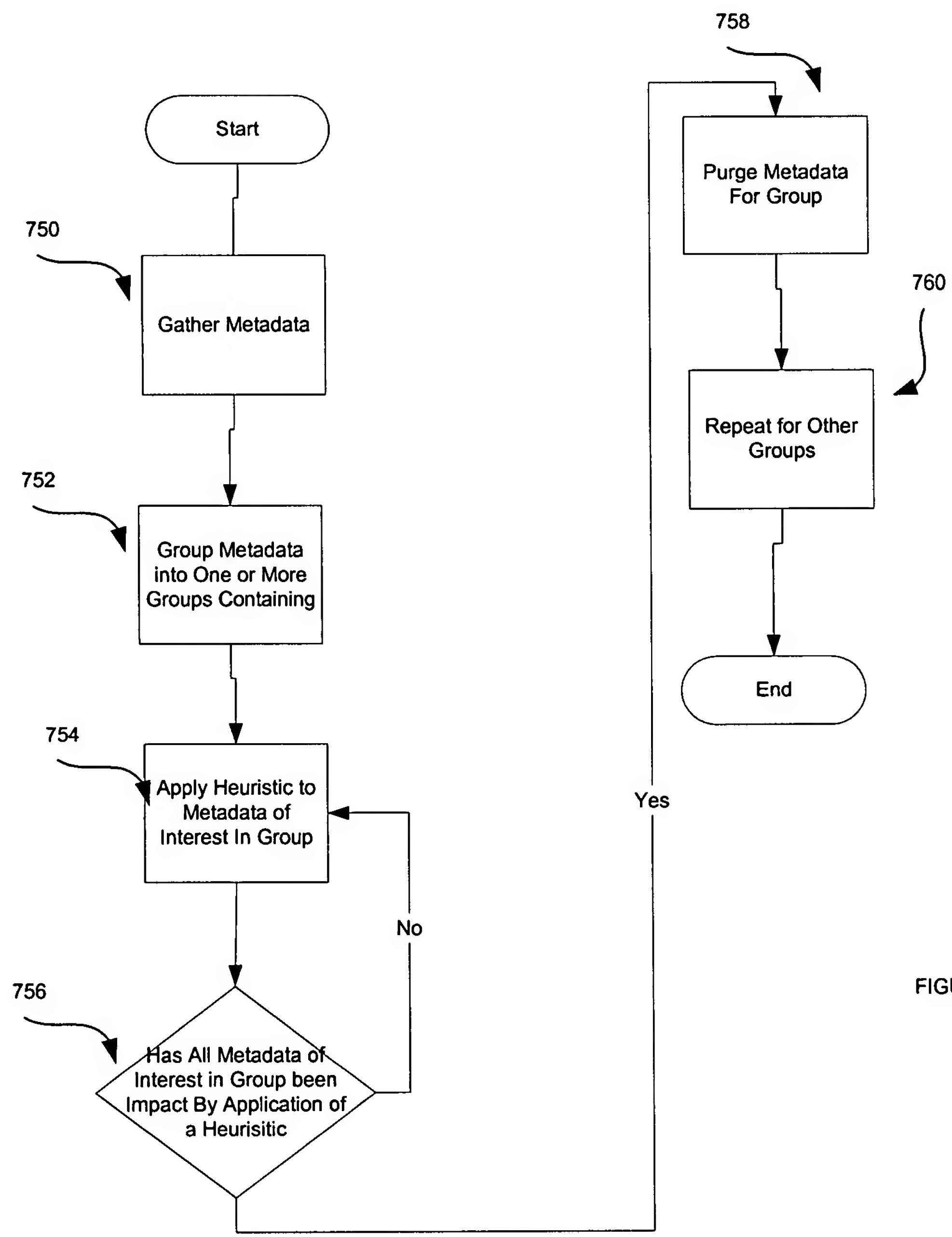
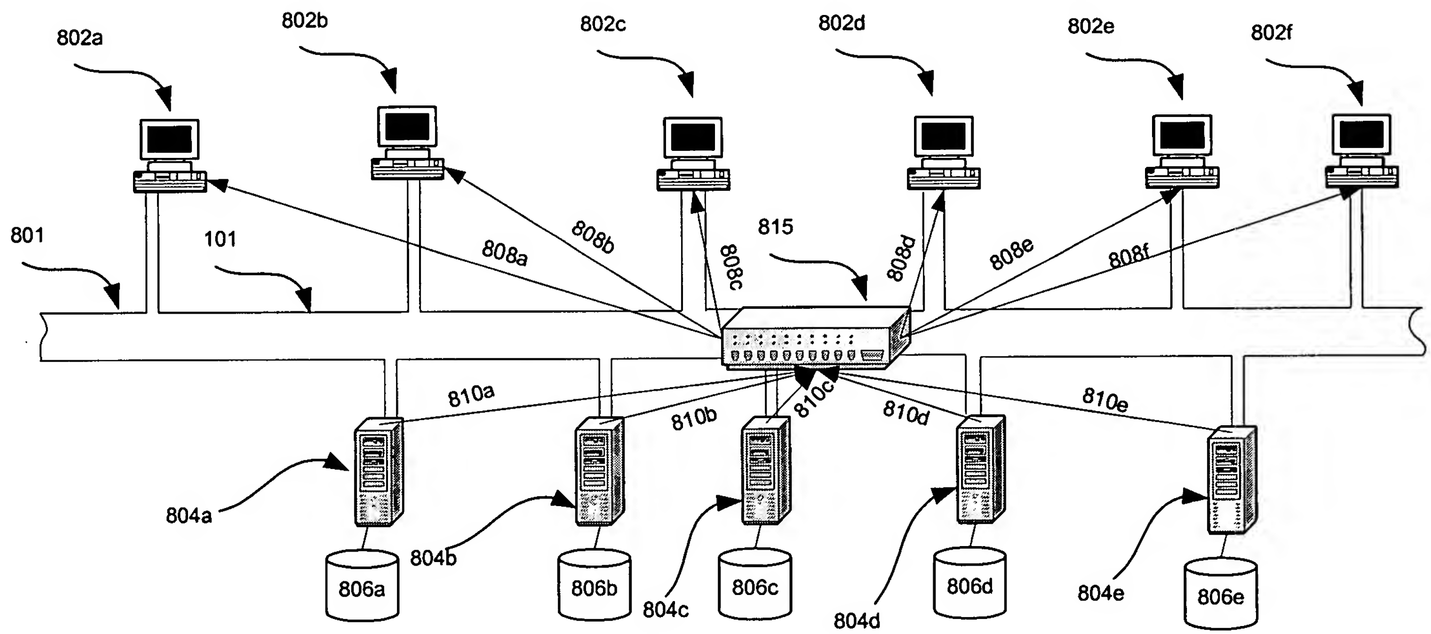
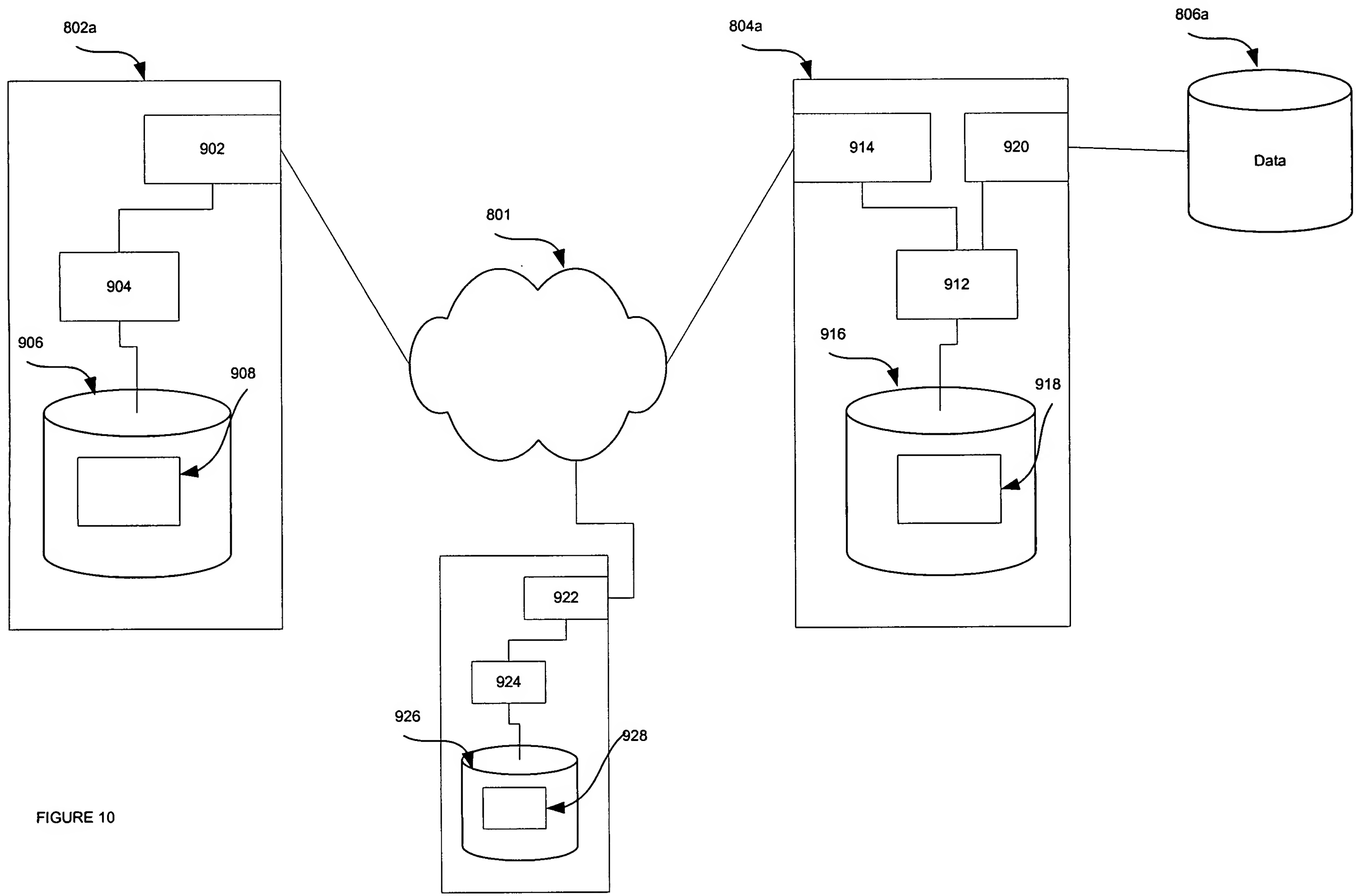

Start
750
Gather Metadata
752
Group Metadata into One or More Groups Containing
754
Apply Heuristic to Metadata of Interest In Group
756
Has All Metadata of Interest in Group been Impact By Application of a Heurisitic
No
Yes
758
Purge Metadata For Group
760
Repeat for Other Groups
End

FIGURE 8



802a
802b
802c
802d
802e
802f
801
101
815
808a
808b
808c
808d
808e
808f
810a
810b
810c
810d
810e
804a
804b
804c
804d
804e
806a
806b
806c
806d
806e

FIGURE 9



802a
902
904
906
908
801
804a
914
920
912
916
918
806a
Data
922
924
926
928

FIGURE 10

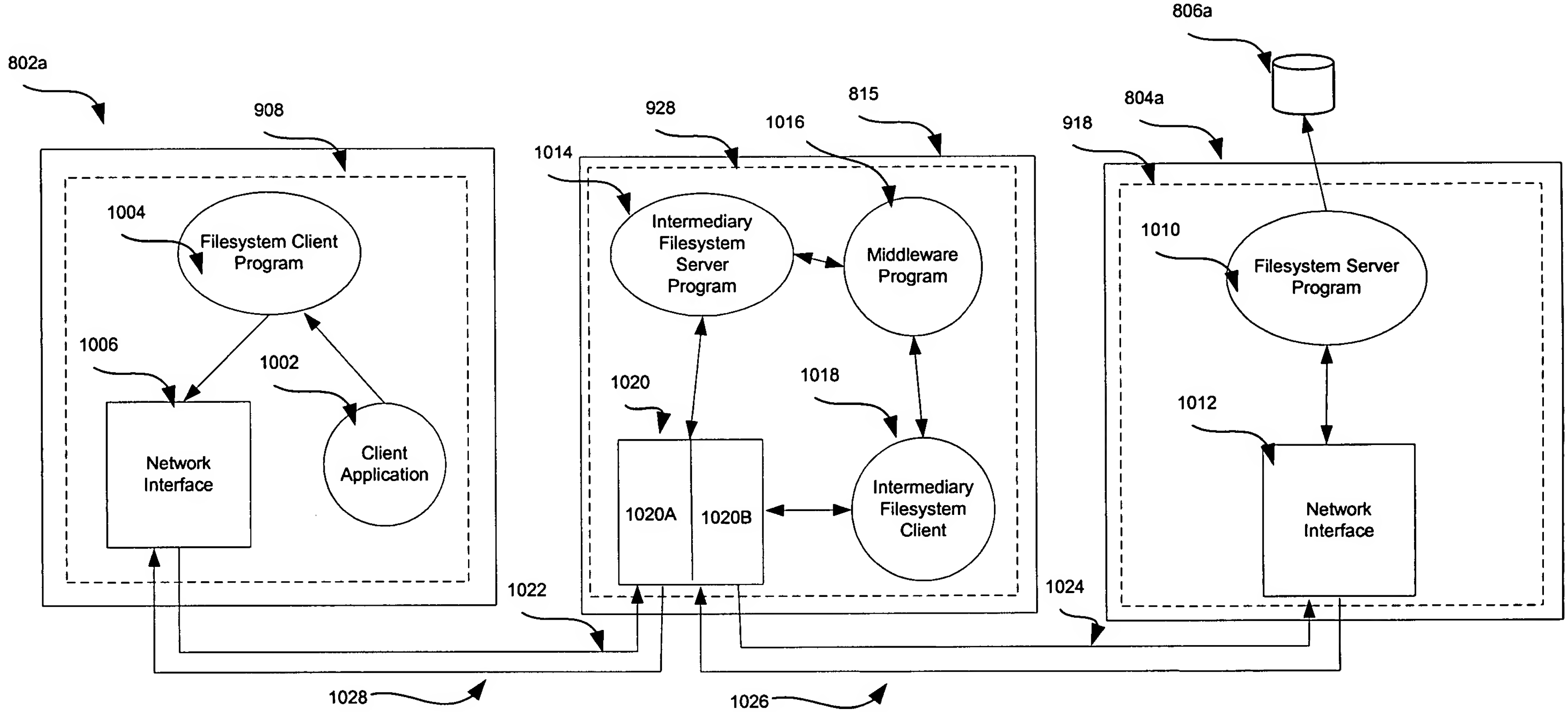

802a
908
1004
Filesystem Client Program
1006
1002
Network Interface
Client Application
928
1014
1016
815
Intermediary Filesystem Server Program
Middleware Program
1020
1018
1020A
1020B
Intermediary Filesystem Client
1022
1028
1026
1024
806a
804a
918
1010
Filesystem Server Program
1012
Network Interface

FIGURE 11

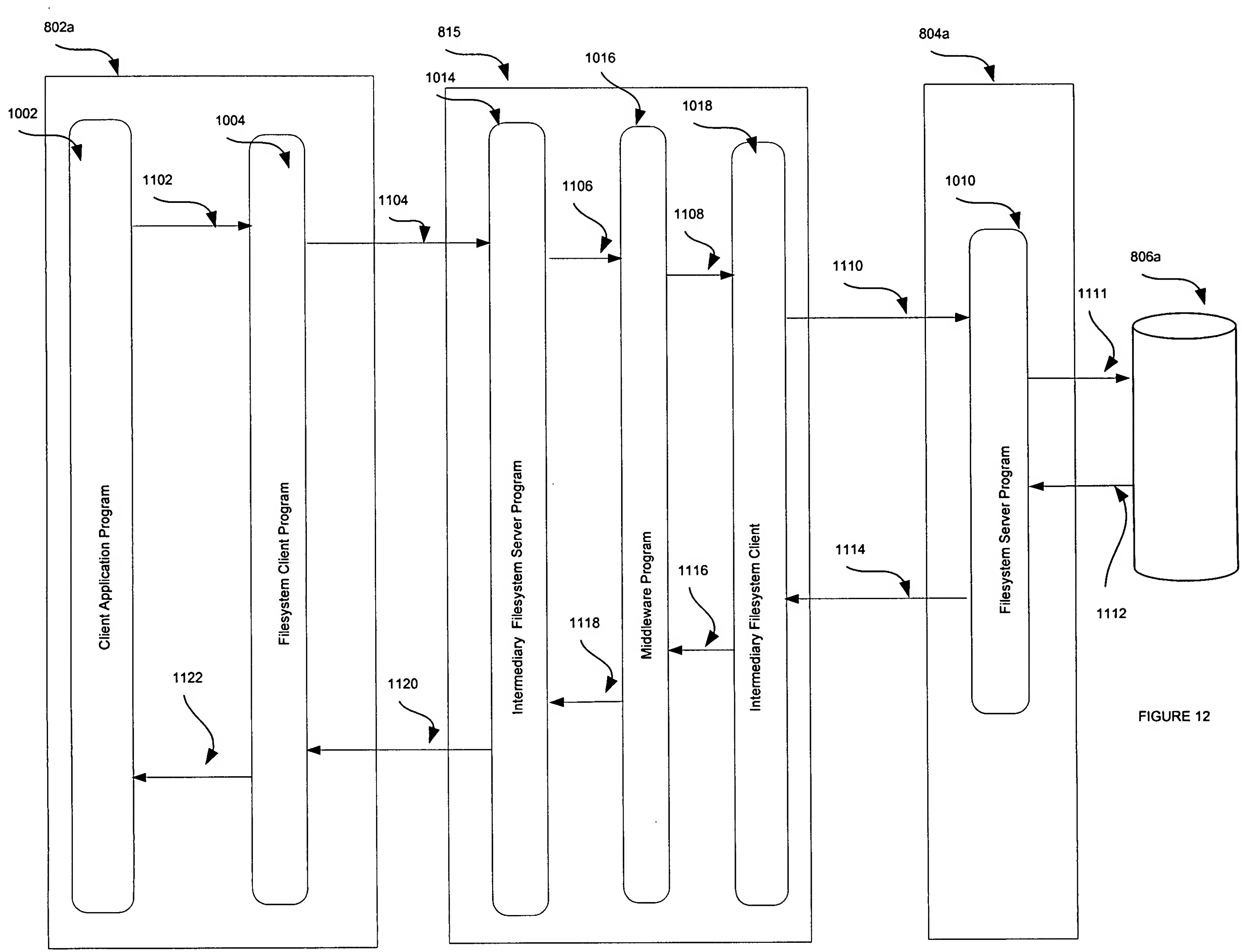

802a
1002
Client Application Program
1102
1004
Filesystem Client Program
1122
1104
1120
815
1014
Intermediary Filesystem Server Program
1106
1118
1016
Middleware Program
1108
1116
1018
Intermediary Filesystem Client
1110
1114
804a
1010
Filesystem Server Program
1111
1112
806a

FIGURE 12

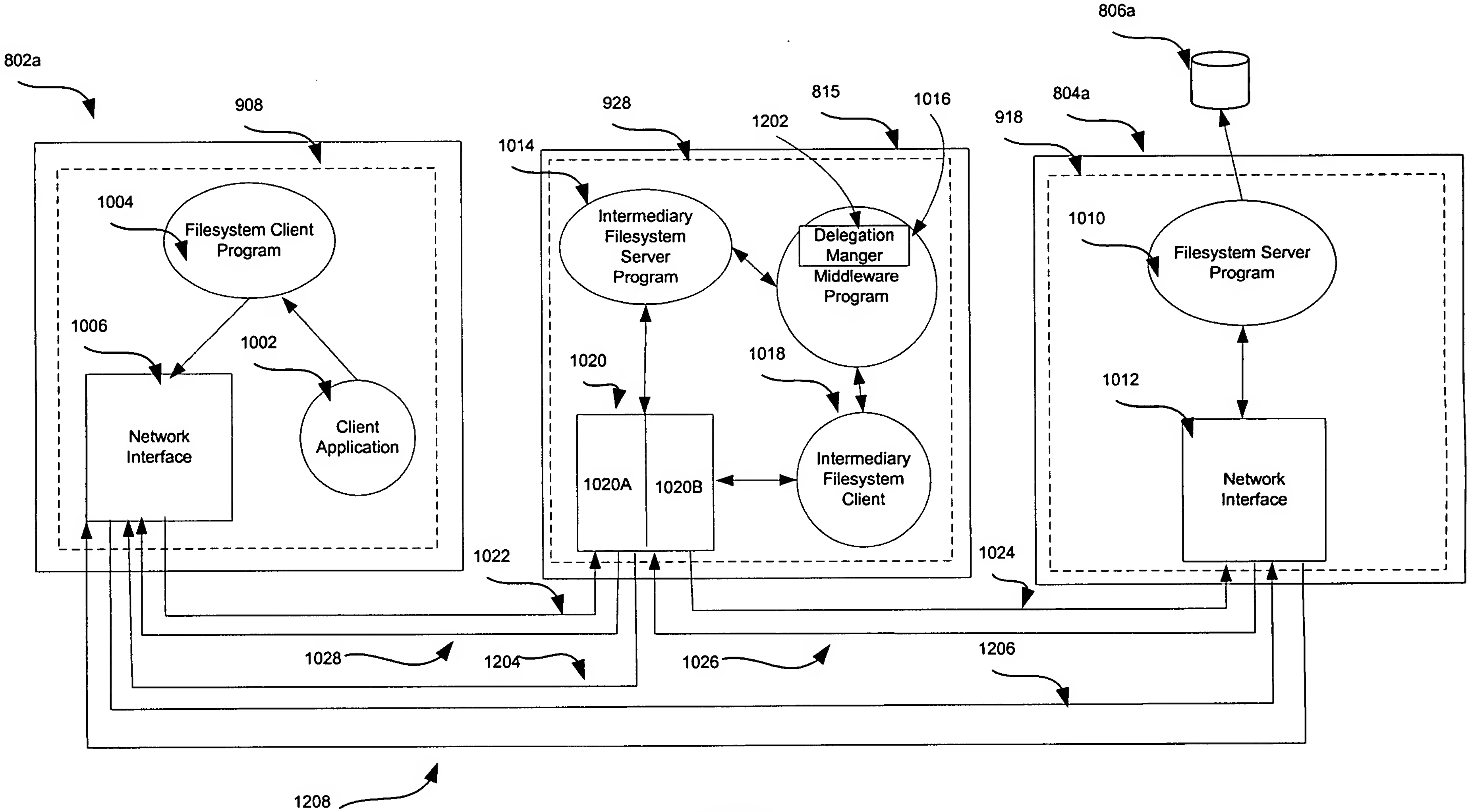

802a
908
1004
Filesystem Client Program
1006
Network Interface
1002
Client Application
1014
928
1202
815
1016
Intermediary Filesystem Server Program
Delegation Manger
Middleware Program
1020
1018
1020A
1020B
Intermediary Filesystem Client
918
804a
806a
1010
Filesystem Server Program
1012
Network Interface
1022
1024
1028
1204
1026
1206
1208

FIGURE 13

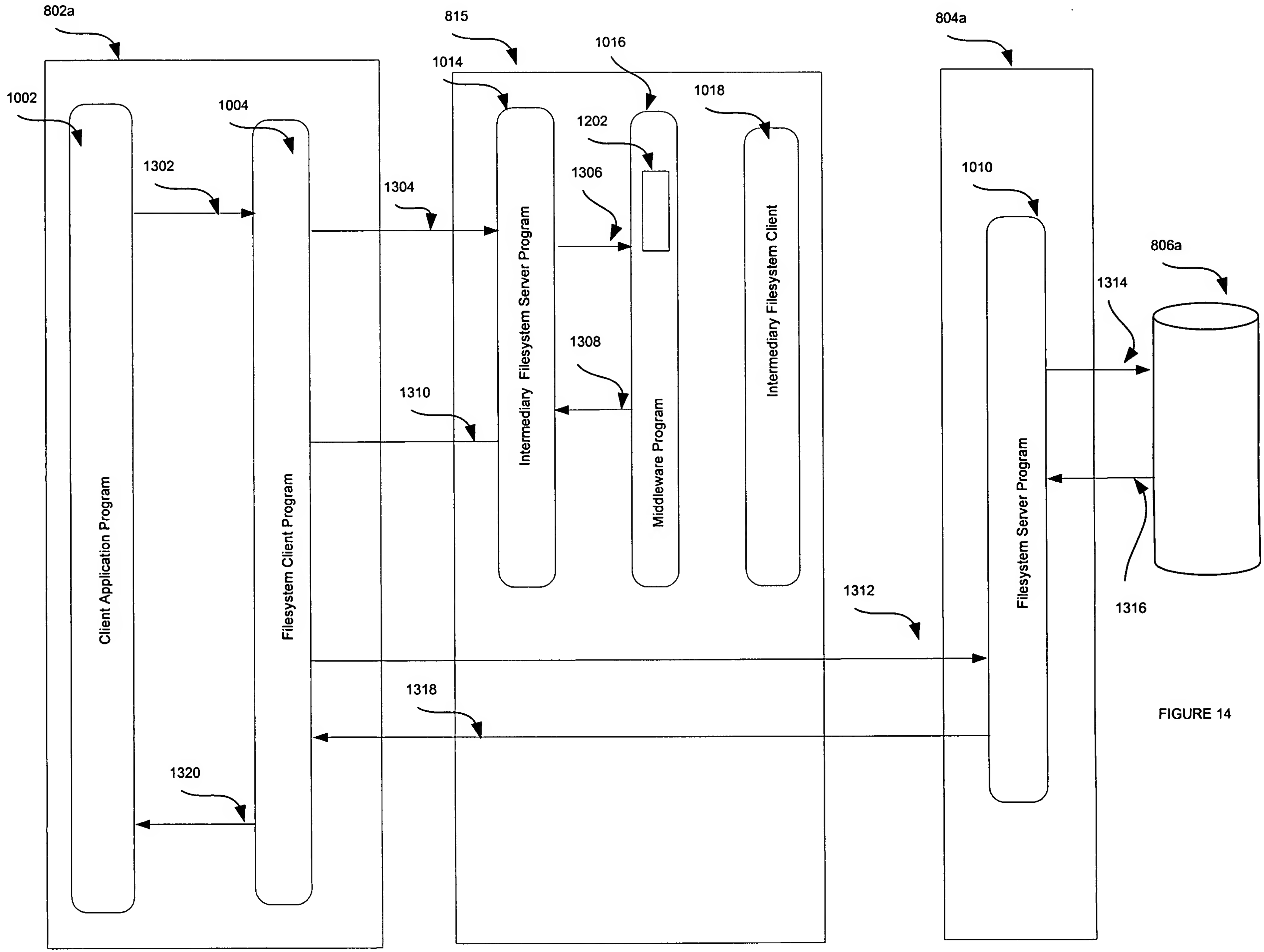

802a
1002
1004
1302
Client Application Program
Filesystem Client Program
1320
815
1014
1016
1018
1202
1304
1306
1308
1310
Intermediary Filesystem Server Program
Middleware Program
Intermediary Filesystem Client
804a
1010
Filesystem Server Program
1312
1318
806a
1314
1316

FIGURE 14

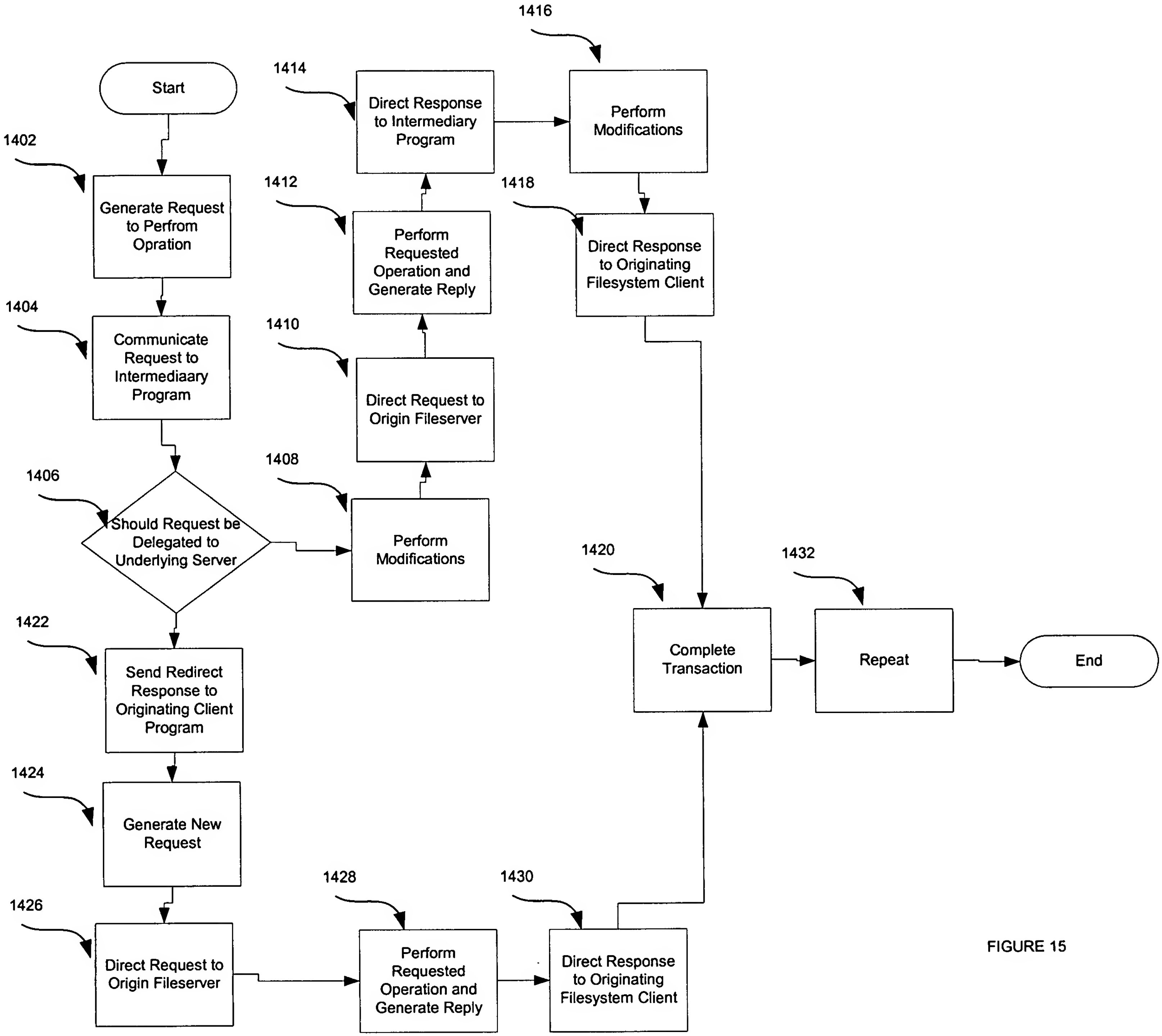
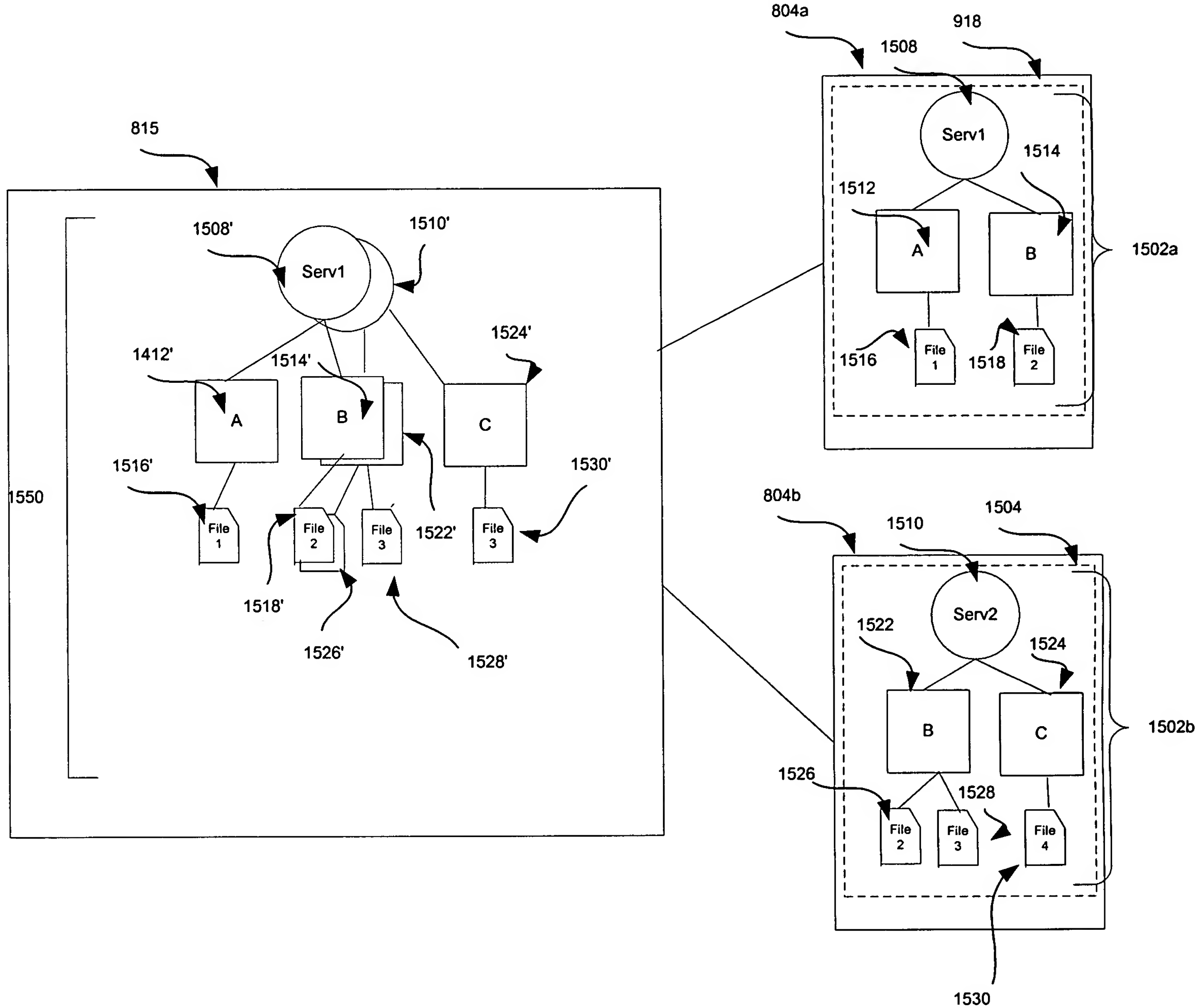
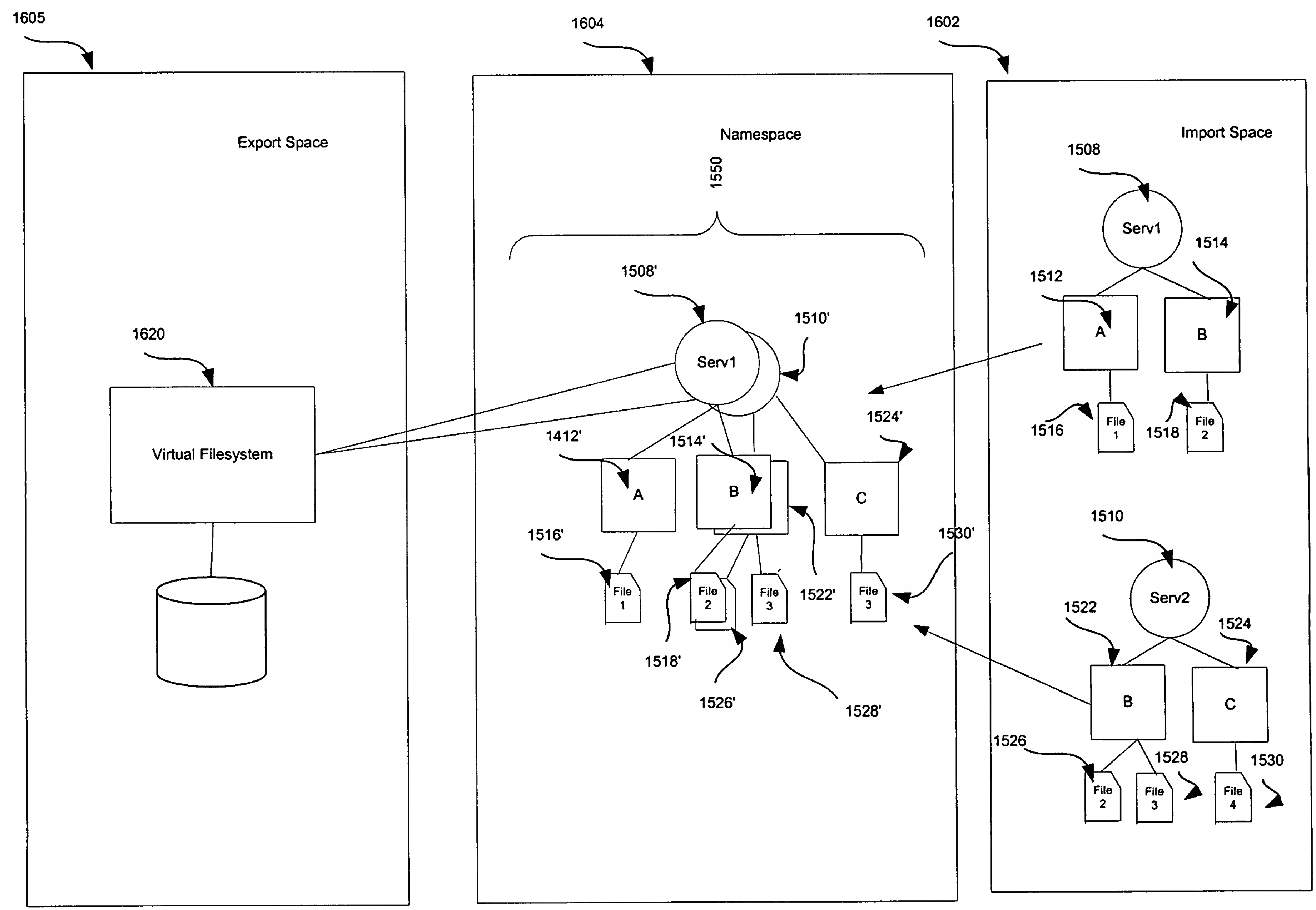

Start
1402
Generate Request to Perfrom Opration
1404
Communicate Request to Intermediaary Program
1406
Should Request be Delegated to Underlying Server
1408
Perform Modifications
1410
Direct Request to Origin Fileserver
1412
Perform Requested Operation and Generate Reply
1414
Direct Response to Intermediary Program
1416
Perform Modifications
1418
Direct Response to Originating Filesystem Client
1420
Complete Transaction
1432
Repeat
End
1422
Send Redirect Response to Originating Client Program
1424
Generate New Request
1426
Direct Request to Origin Fileserver
1428
Perform Requested Operation and Generate Reply
1430
Direct Response to Originating Filesystem Client

FIGURE 15



815
1508'
Serv1
1510'
1412'
1514'
1524'
A
B
C
1516'
1530'
1550
File 1
File 2
File 3
File 3
1522'
1518'
1526'
1528'
804a
918
1508
Serv1
1514
1512
A
B
1502a
1516
File 1
1518
File 2
804b
1504
1510
Serv2
1522
1524
B
C
1502b
1526
1528
File 2
File 3
File 4
1530

FIGURE 16



1605
1604
1602
Export Space
Namespace
Import Space
1550
1620
Virtual Filesystem
1508'
1510'
Serv1
1412'
1514'
1524'
A
B
C
1516'
1522'
1530'
File 1
File 2
File 3
File 3
1518'
1526'
1528'
1508
Serv1
1512
1514
A
B
1516
1518
File 1
File 2
1510
Serv2
1522
1524
B
C
1526
1528
1530
File 2
File 3
File 4

Figure 17

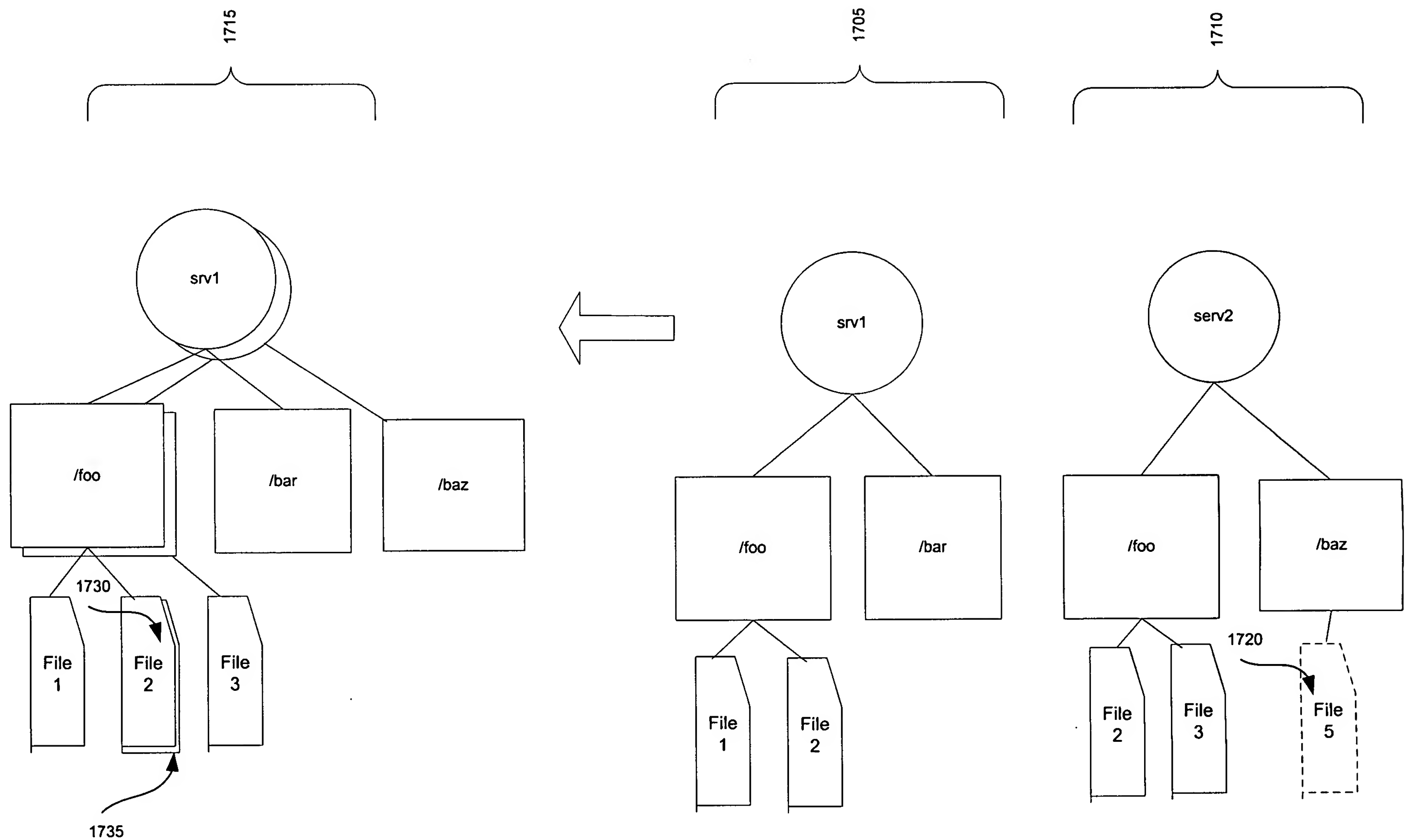

1715
srv1
/foo
/bar
/baz
1730
File 1
File 2
File 3
1735
1705
srv1
/foo
/bar
File 1
File 2
1710
serv2
/foo
/baz
1720
File 2
File 3
File 5

FIGURE 18

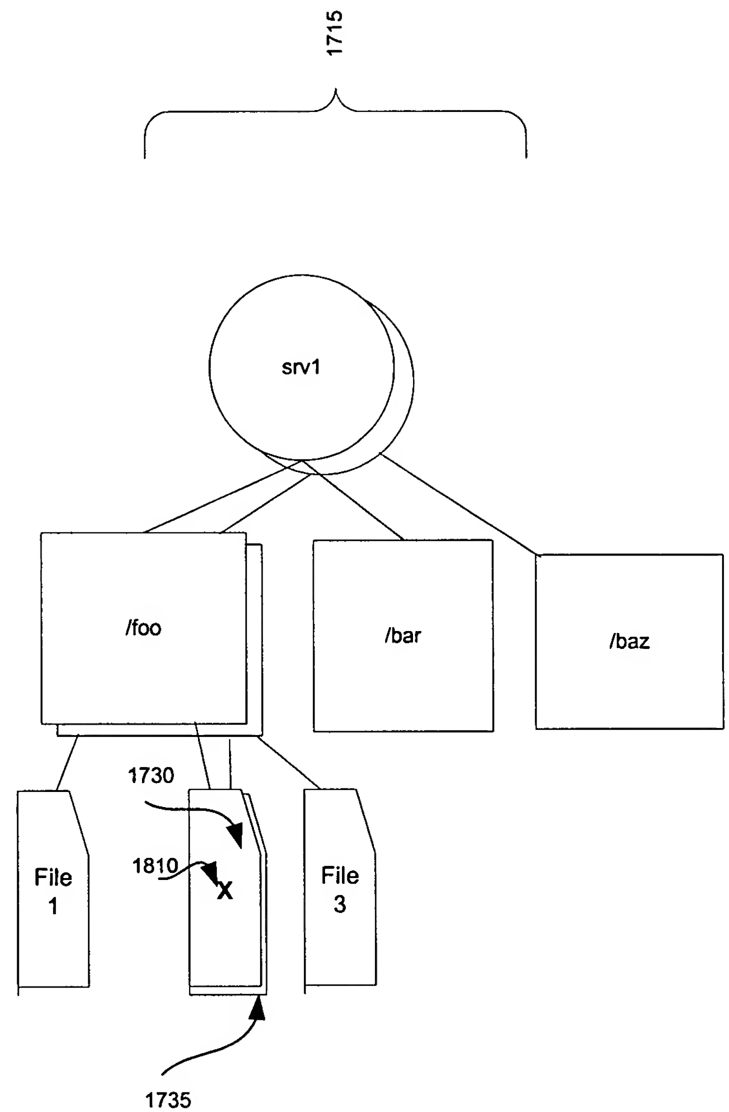


Figure 19

```
{{{
/* This module implements "write-through" semantics:
   First, the operation is attempted in the topbase,
   If the file/dir doesn't exist in the topbase, then
   it is attempted in the bottombases recursively.  We
   consider only pairwise layers;  it is understood that
   the stack is arbitrarily deep, and upon each iteration
   through the stack the previous bottombase becomes the
   new topbase.

   Whiteouts:
   if a file exists in both layers:
     if it is removed: remove from top, create a whiteout to hide bottom
   if a file exits on top and not on bottom layer:
     if it is removed: remove from top
   if a file exists in bottom and not on top layer:
     if it is removed: remove from bottom
   if a file is whiteout on top and it exists in bottom:
   if a file is whiteout on top and it exists in bottom:
     if it is removed: do nothing
     if it is created: remove whiteout and create one on top
     if it is to be accessed: FAIL


   When an operation involves 2 file names:
   - rename(from,to) gets called only if from and to are in this namespace.
   - symlink(from,to) gets called only if from is in this namespace
           (to may or may not be in the name space)
   - link(from,to) gets called only if from and to are in this namespace
*/
```

FIGURE 20

```
/* GROUP 1: ========================================
  Operations on file that must exist.
  getattr
  readlink
  chmod
  chown
  truncate
  utime
  read

  Semantics:
  fcn (path, args)
  {
    GetTopPathState(path, NULL, &topExists, &isWhiteOut, &topPath);
    if (topExists)
    {
      // exists in top layer, use it
      return lowerFcn(topPath, args);
    }
    else
    {
      if (isWhiteOut)
        {
          // it's white out on top, FAIL
          return - ENOENT;
        }
        else
        {
          GetBottomPathState(path, NULL, &bottomExists, NULL, &bottomPath);
          if (bottomExists)
          {
            // doesn't exist on top, exists on bottom, use it
         return lowerFcn(bottomPath, args);
          }
          else
          {
            // doesn't exist on top or bottom
            return - ENOENT;
          }
        }
    }
  }
*/
```

FIGURE 21

```
/* GROUP 2: ===========================================
 Operations on file that must not exist. Operation create the file.
 mknod
 mkdir

 Semantics:
 fcn (path, args)
 {
  GetTopPathState(path, &topMatchLen, &topExists, &isWhiteOut, &topPath);
  if (topExists)
  {
   // exists in top layer, FAIL
       return EEXIST;
  }
  else
  {
   if (isWhiteOut)
       {
         // it's white out on top, remove without and perform operation
         DelWhiteOut(topPath);
         return lowerFcn(topPath, args);
       }
       else
       {
         GetBottomPathState(path, &bottomMatchLen, &bottomExists, NULL, &bottomPath);
         if (bottomExists)
         {
          // exists on bottom, FAIL
          return EEXIST;
         }
         else
         {
          // doesn't exist on top or bottom, create file on layer with deeper match
          if (topMatchLen >= bottomMatchLen)
          {
            return lowerFcn(topPath, args);
          }
          else
          {
            return lowerFcn(bottomPath, args);
          }
         }
       }
  }
 }
*/
```

FIGURE 22

```
/* GROUP 3: ==========================================
  Operations on file if it exists, file created if it doesn't.
  open
  write

  Semantics:
  fcn (path, args)
  {
    GetTopPathState(path, &topMatchLen, &topExists, &isWhiteOut, &topPath);
    if (topExists)
    {
      // exists in top layer, use it
          return lowerFcn(topPath, args);
    }
    else
    {
      if (isWhiteOut)
          {
            // it's white out on top, remove without and perform operation
            DelWhiteOut(topPath);
            return lowerFcn(topPath, args);
          }
          else
          {
            GetBottomPathState(path, &bottomMatchLen, &bottomExists, NULL, &bottomPath);
            if (bottomExists)
            {
              // exists on bottom, use it
              return lowerFcn(bottomPath, args);
            }
            else
            {
              // doesn't exist on top or bottom, create file on layer with deeper match
              if (topMatchLen >= bottomMatchLen)
              {
                return lowerFcn(topPath, args);
              }
              else
              {
                return lowerFcn(bottomPath, args);
              }
            }
          }
    }
  }
*/
```

FIGURE 23

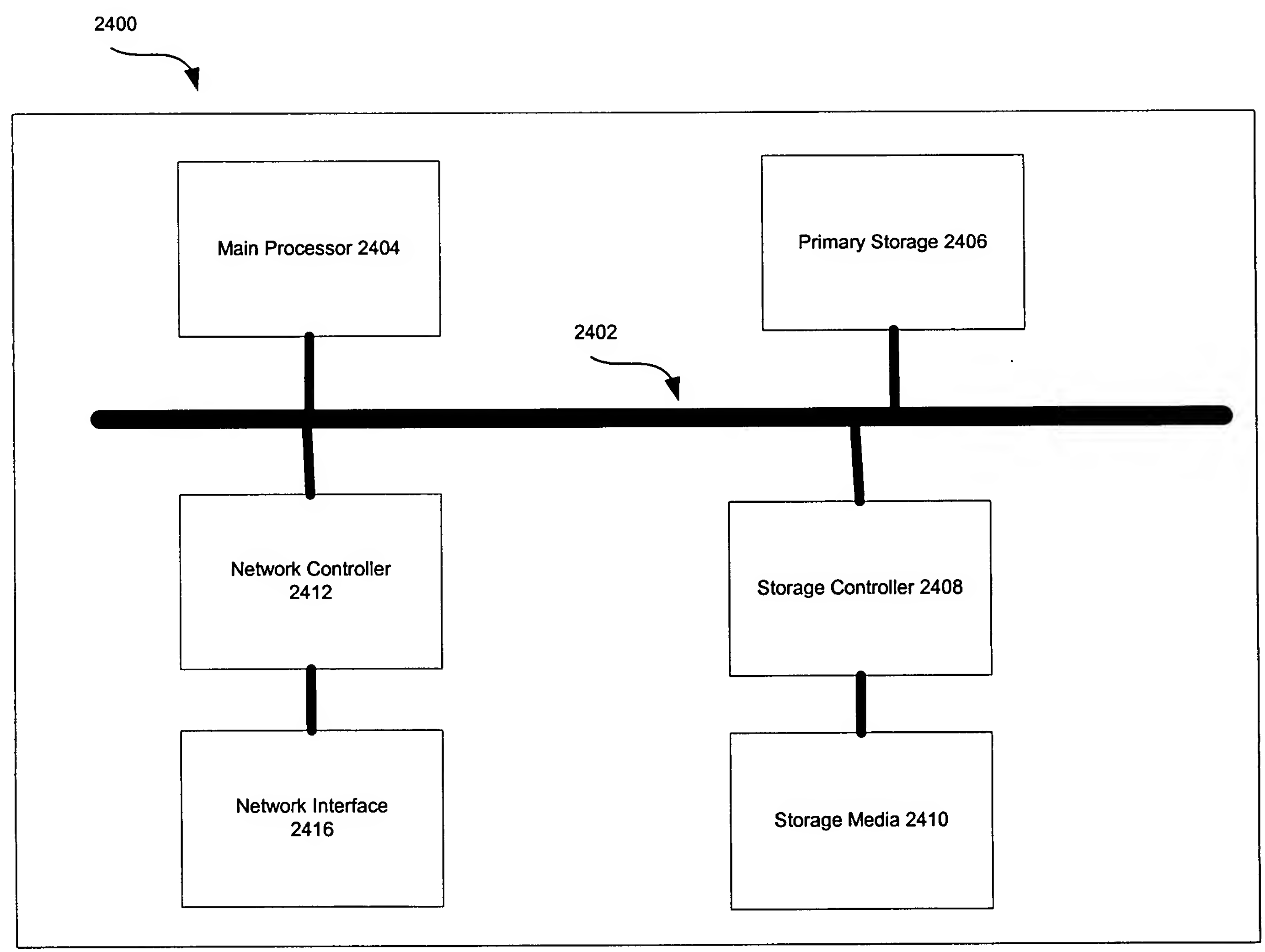

2400
Main Processor 2404
Primary Storage 2406
2402
Network Controller
2412
Storage Controller 2408
Network Interface
2416
Storage Media 2410

FIGURE 24